LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-463475

Energy-efficient specialization of functional units in a Coarse-Grained Reconfigurable Array

B. Van Essen, R. Panda, A. Wood, C. Ebeling, S. Hauck

December 2, 2010

Nineteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays
Monterey, CA, United States
February 27, 2010 through March 1, 2010

Disclaimer

Energy-Efficient Specialization of Functional Units in a Coarse-Grained Reconfigurable Array

Brian Van Essen[†‡*] Robin Panda[§] Aaron Wood[§]
Carl Ebeling[†] Scott Hauck[§]

[†] Dept. of Computer Science and Engineering and [§] Dept. of Electrical Engineering
University of Washington, Seattle, WA 98195
[†] {vanessen, ebeling}@cs.washington.edu [§] {robin, arw28, hauck}@ee.washington.edu

[‡] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550
[‡] vanessen1@llnl.gov

ABSTRACT

Functional units provide the backbone of any spatial accelerator by providing the computing resources. The desire for having rich and expensive functional units is in tension with producing a regular and energy-efficient computing fabric. This paper explores the design trade-off between complex, universal functional units and simpler, limited functional units.

We show that a modest amount of specialization reduces the area-delay-energy product of an optimized architecture to 0.86× a baseline architecture. Furthermore, we provide a design guideline that allows an architect to customize the contents of the computing fabric just by examining the profile of benchmarks within the application domains.

Categories and Subject Descriptors

B.2.1 [**Arithmetic and Logic Structures**]: Design Styles; C.1.1 [**Processor Architectures**]: Single Data Stream Architectures

General Terms

Design, Performance

Keywords

CGRA, energy-efficiency, functional units, architecture

1. INTRODUCTION

Functional units are the core of compute-intensive spatial accelerators. They perform the computation of interest with support from local storage and communication structures. Ideally, the functional units will provide rich functionality, supporting operations ranging from simple addition, to fused multiply-adds, to advanced transcendental functions and domain specific operations like add-compare-select. However, the total opportunity cost to support the more complex operations is a function of the cost of the hardware, the rate of occurrence of the operation in the application domain, and the inefficiency of emulating the operation with simpler operators. Examples of operations that are typically emulated in spatial accelerators are division and trigonometric functions, which can be solved using table-lookup based algorithms [1] and the CORDIC algorithm [2].

One reason to avoid having direct hardware support for complex operations in a tiled architecture like a Coarse-Grained Reconfigurable Array (CGRA) is that the expensive hardware will typically need to be replicated in some or all of the architecture's tiles. Tiled architecture are designed such that their tiles are either homogenous or heterogenous. Homogenous architectures are simpler to design but heterogeneous architectures can be more efficient. Generally, CGRAs try to support a rich set of operations with the smallest possible set of hardware devices.

[*]This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

FPGA'11, February 27–March 1, 2011, Monterey, California, USA.

2. BACKGROUND

This work builds upon the Mosaic research infrastructure and optimized architectures that were designed in [3] and [4]. Furthermore, this work uses the same suite of benchmarks and 65nm process for the architecture's physical models as [4].

2.1 Architecture

The Mosaic CGRA architectures are a class of statically scheduled coarse-grained reconfigurable arrays. They are designed to exploit loop-level parallelism in an application's computationally-intensive inner loops (*i.e.* kernels) in an energy-efficient manner. The architectures are dynamically reconfigured, so that they time-multiplex their functional units and interconnect on a cycle-by-cycle basis. Like many CGRAs ([5], [6], [7]), the Mosaic architecture fits in the design space between FPGAs and VLIW processors, with similarities to a word-wide FPGA and a 2-D VLIW.

The Mosaic CGRA architecture (shown in [4]) is a cluster-based architecture that is arranged as a 2-D grid. To minimize the design complexity, the clusters are homogenous in the set of functional units that are supported. All of the clusters have four 32-bit processing elements. The 32-bit datapath in each cluster also has two large rotating register files for long-term storage, two distributed registers, one or two data memories, and connections to the grid interconnect. The 32-bit data path is complemented by a 1-bit control path that handles predicate generation and evaluation.

2.2 Related Work

Few other research efforts have evaluated the energy advantages and tradeoffs of architectural features in CGRAs. By and large

Benchmark	Category	32-bit Logic & Comparison	Arithmetic Ops. Simple	Arithmetic Ops. Complex	Select	Arith. & Logical Shifts	1-bit Logic & Comparison
FIR	Complex	0.7	0.7	47.1	47.1	0.7	3.7
FIR (Banked)	Complex	3.6	23.2	42.9	24.1	0.9	5.4
Convolution	Select	3.4	1.1	27.8	63.4	0.6	3.7
Dense matrix multiply	Select	5.5	0.9	14.6	61.2	14.6	3.2
Motion Estimation	Simple	40.9	18.5	7.2	18.1	3.9	11.5
Smith-Waterman	Simple	39.8	21.0	0.2	36.8	0.0	2.2
K-Means Clustering	Simple	34.5	25.3	0.0	33.5	0.0	6.8
CORDIC	Simple	15.0	26.4	0.0	39.3	7.1	12.1
PET Event Detection	Simple	33.8	8.9	2.9	46.6	5.8	2.0
Matched filter	Balanced	9.5	7.1	21.4	22.2	25.4	14.3
Average		24.4	13.1	11.0	41.0	5.0	5.5

Table 1: Frequency of dynamic operations, reported as percentage, in the benchmark suite.

these efforts have focused on ad-hoc system analysis that examine multiple architecture features simultaneously. Kim et al. [8] looked at the tradeoffs for what they call primitive versus critical functional unit resources in a MorphoSys-style CGRA. Their study found that pipelining and sharing multipliers with other processing elements substantially reduced the area and delay of the architecture. These results align well with the results presented here, although our results started out with a pipelined multiplier and were conducted on a much larger scale. Wilton et al. [9] explore the connectivity requirements for point-to-point interconnects in an ADRES architecture with a heterogenous mix of functional units, when optimizing for area and performance. Others such as [10] were smaller evaluations that confirmed benefits from state of the art design practices, such as clustering multiple functional units together.

3. EVALUATING OPERATOR FREQUENCY

This experiment uses the benchmark suite as detailed in [3] and [4]. Table 1 shows the breakdown of operations within the benchmark suite, as well as an average frequency for each class of operation. Note that simple arithmetic operations are addition, subtraction, and negation, while multiplication and fused multiply-add or multiply-subtract operations are complex operations. Table 1 also shows how applications can be broadly categorized by the dominant type of operations. Note that complex arithmetic was given special priority with a threshold of only 40%, due to its relative importance in the application domain and complexity of the hardware.

- Balanced - relatively even mix of all operation types
- Simple-dominant - $\geq$ 60% operations are simple arithmetic or logic / comparison operations
- Complex-dominant - $\geq$ 40% operations are complex arithmetic
- Select-dominant - $\geq$ 60% operations are select operations

The profile of the benchmark applications shows two key things: select operations are extremely common, while complex multiply or multiply-add operations rarely exceed 25% (and never 50%) of operations in the suite. Select operations provide dynamic data steering within the statically scheduled CGRA, typically staging data for subsequent computation. This makes it attractive to co-locate select hardware with other functional units, such as the ALU.

4. DESIGNING THE FUNCTIONAL UNITS

Tiled spatial accelerators typically eschew embedding the more complex and esoteric functional units, in favor of a simpler repeated tile, focusing on a range that spans simple adders to fused multiply-add units. In this experiment we explore the following primitive and compound functional units for the word-wide datapath. Note that the single-bit datapath uses a 3-input lookup table for its functional units.

- ALU - arithmetic and logic unit, with support for select
- Shifter - logarithmic funnel shifter
- MADD - 2-cycle fused multiply-add
- S-ALU - compound unit with shifter, ALU, and select
- Universal - compound unit with MADD, shifter, ALU, and select

4.1 Compound functional units

The S-ALU and universal compound functional units combine multiple primitive functional units into a single logical group. It is notable that while a compound functional unit could support multiple concurrent operations internally, it lacks the input and output ports necessary to supply and produce multiple sets of operands and results. Sharing the input and output ports mitigates the need to increase the size of the cluster's crossbar as more functionality is added to each functional unit. The cost for adding a port to the crossbar is approximately the same as the hardware for an ALU. In addition to the cost of adding a port to the crossbar, each input and output of the functional unit requires some peripheral hardware in the processing element (PE) to stage operands and results.

By treating compound FUs as one logical device, the placement tool will only map a single operation onto the functional unit per cycle. This ensures that an architecture maintains the same number of concurrent operations per cluster when mixing primitive and compound functional units. Two other advantages of maintaining the same number of concurrent operations per cluster are 1) the external support resources in the cluster do not have to be scaled as capabilities are added to each functional unit, and 2) it is easier to test and make comparisons between two architectural variants.

4.2 Comparison of Functional Units

To evaluate the tradeoff between flexibility and overhead for the functional units we examine several of their characteristics. The area and energy metrics for the each type of processing element (*i.e* functional unit plus word-wide peripheral logic) and their associated crossbar I/O ports are presented in Table 2. Note that the configuration energy includes both the static and dynamic energy of the configuration SRAM and associated logic, since the dynamic energy consumed per clock cycle can be precomputed. The peripheral resources include the local register files, input retiming registers, and multiplexers.

One advantage of using compound functional units instead of a larger number of primitive functional units is that it minimizes the

Processing Element	Area	Static Energy	Config. Energy	Datapath Ports (I/O)
ALU	17754	12.2	519.2	2/1
Shifter	17703	12.3	519.2	2/1
MADD	30754	16.0	631.4	3/1
S-ALU	19177	12.4	531.7	2/1
Universal	38357	19.5	814.4	4/1

Table 2: Characteristics for each type of processing elements (functional unit plus peripheral logic) and crossbar I/O ports. Area is reported in μm^2. Static and configuration energy was computed for a clock period of 1.93ns and are reported for $fJ/cycle$.

number of output ports and peripheral resources required. Since the functional units consume more values than they produce, there are fewer inputs to the crossbar than outputs from it, and thus the crossbar is not square. Therefore, the cost to add an output port to the functional unit (or attaching another device to the crossbar) is significantly more expensive than adding an input port, primarily due to the high number of capacitive loads within the crossbar.

5. EXPERIMENTS

To explore the impact of specializing the functional units in a cluster we test several clusters built with different functional units. The baseline architecture has four universal functional units per cluster and the optimized storage design from [4], with a private rotating register file in each functional unit and one cluster-wide large rotating register file. Each test in this experiment replaced some of those four functional units with a more specialized device. Two design considerations that were followed during this experiment were 1) each cluster could perform all supported operations, and 2) the number of concurrent operations per cluster remained constant.

To specialize the functional units we looked at the frequency of operations in the benchmark suite. The first optimization is to reduce the total number of multipliers within the architecture because they are the most expensive units, require the most input ports, and multiplication and MADD operations only make up $\sim$11% of the dynamic operation mix. As the number of multipliers is reduced, architectures will have a small number of either Universal FUs or MADDs, and the remaining functional units will be either be S-ALUs or ALUs, which are abbreviated as `U`, `M`, `S`, and `A`, respectively. The set of permutations that we explored in this paper are designated as architectures: A - `4U`, B - `3Ux1S`, D `2Ux2S`, E - `2Ux1Sx1A`, F - `2Ux2A`, G - `1Ux3S`, and K - `1Mx3S`.

To test the impact of specializing the functional units, each application in the benchmark suite was mapped to each architectural variant multiple times with different random placement seeds. The target architecture for each application was sized so that the application's critical resource consumed approximately 80% of the architecture's resources. We used 12 placement seeds per application, although FIR and convolution were tested with 20 seeds because they showed a higher variability in quality of placement. After simulating each application to architecture mapping, the area-delay-energy (ADE) product for each tests was calculated and used to select the random seed that produced the best ADE results.

6. RESULTS

Specialization of the functional units involves three key principles: 1) stripping away expensive and underutilized resources, 2) avoiding overly specialized hardware, and 3) creating functional units with rich functionality when the extra logic is inexpensive (*i.e.* maximizing the utility of fixed resources). The effects of each of these principles is explored in the following three sections as the architecture moves from a general to a specialized fabric design.

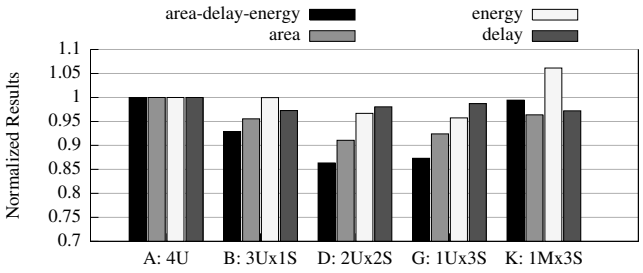

Figure 1: Average area-delay-energy product for architectures that are specialized with a mix of universal, S-ALU, and MADD FUs functional units.

6.1 Paring down underutilized resources

Figure 1 shows the area-delay-energy product, averaged across all benchmarks, for several architecture designs. The delay metric is the execution time averaged across all applications, and the total energy reported includes dynamic, static, configuration, and clock energy. This section focuses on the first four columns, which shows the trends as the MADD units are removed from individual functional units in architectures A, B, D, and G. Specializing the functional units by removing superfluous MADD devices reduces the area-delay-energy product by as much as $0.86\times$ the area-delay-energy of the baseline architecture. The best specialized design, D, has 2 Universal FUs and 2 S-ALU FUs, versus a baseline with 4 Universal FUs.

As the designs go from having 2 to 1 Universal FU per cluster in architectures D and G, the number of clusters required goes up by 6.5%, which gives architecture D its performance advantage. This happens because, as we reduce the number of MADD devices, the complex-dominant (45.0% multiplication) and select-dominant (21.2% multiplication) applications become resource-starved, and require more clusters to have enough multipliers.

6.2 Avoiding excessive specialization

Given the benefits of paring down underutilized resources, it may seem obvious to replace at least one universal functional unit with a dedicated MADD FU, as tested by architecture K. Not only is the MADD functional unit smaller than the Universal FU, it also requires one less crossbar port. It turns out that architecture K is overly specialized with the MADD FU and performs worse, in terms of overall area-delay-energy product and total energy consumed, than all other specialized architectures, and only marginally better than the general design with four Universal FUs as shown in Figure 1. While architecture K required 7% more clusters than G, the 11% increase in energy is partly due to G being able to co-locate instructions on a single functional unit better than K. This allows sequences of operations to stay in the same processing element instead of having to traverse the cluster's crossbar.

6.3 Exploiting fixed resource costs

The third design principle we mentioned earlier is to make functional units as rich as possible when the additional functionality is cheap. Since shift operations are relatively rare in our benchmarks, it might make sense to reduce some or all of the S-ALUs to pure

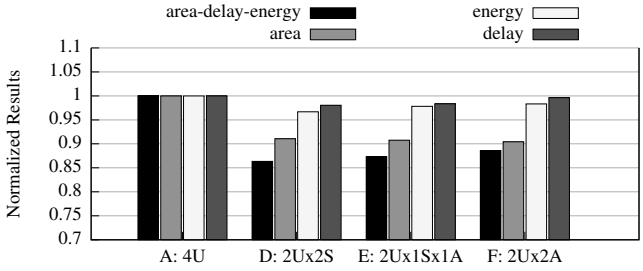

Figure 2: Average area-delay-energy product for architectures with 2 universal FUs and a mix of S-ALU and ALU functional units, with architecture A as reference.

ALUs. However, when we look at Table 2, we see that the shifter is relatively cheap, and that an ALU is only 0.93× smaller than a S-ALU when the peripheral logic and crossbar connections are also factored in.

Figure 2 compares architectures with 2 Universal FUs, and a mix of S-ALUs and ALUs. As can be seen, the differences between the architectures D, E, and F are small, though as we convert from S-ALUs to ALUs we slightly decrease the area, and slightly increase the power and delay. Therefore, it makes sense to use the S-ALU to maximize the utility of both the peripheral logic and crossbar ports. The other advantage of building richer functional units is an increased opportunity for spatial locality when executing a sequence of operations that requires a diverse set of operators.

7. PREDICTING THE MIX OF FUNCTIONAL UNITS

It is valuable to be able to predict a good functional unit mix for different application domains by examining the characteristics of benchmarks in that application domain, instead of using extensive testing. We present a set of guidelines that combines the three principles for specializing the functional units (Section 6), and other constraints discussed in Section 4. Aside from their applicability to the Mosaic CGRA, these guideline can be applied to other CGRAs and tile-based spatial accelerators.

1. Every tile must support all operations.
2. Remove expensive and underutilized hardware: strip away MADD devices where possible.
3. Avoid over specialization, which can lead to under-utilization and poor spatial locality due to a lack of co-location.
4. Make FUs support rich functionality when it is inexpensive to do so: avoid standalone ALUs since a S-ALU is only marginally more expensive and can make better use of peripheral logic and crossbar resources.
5. Provide functional diversity within a tile to allow collocation and simultaneous execution of operations on expensive resources.

Validating the effectiveness of these guidelines is beyond the scope of this paper, but a preliminary evaluation is shown for the simple-dominant application category presented in Table 1. For the simple-dominant applications, our guidelines predict that G is the right architecture, because there is a much smaller percentage of complex operations, only 2.1% on average and a peak of 7.2%. Experimental results (omitted here due to space restrictions) showed that the predicted architecture G was within 2% of the best architecture, which took advantage of the fact that the number of shift operations is significantly smaller (only 3.4%) than for the balanced-dominant and select-dominant categories, and thus had fewer shifters.

8. CONCLUSIONS

Specializing the functional units within a CGRAs tile can improve the architecture's area-delay-energy product by 0.86× just by paring down infrequently used hardware devices. Specifically, multiply and multiply-add operations are expensive and do not dominate all applications within the benchmark suite, thus requiring only one to two out of the four functional units to support them. While shift operations are also infrequent, they are relatively inexpensive to implement. More importantly, they do not require additional ports from the crossbar beyond what is required for an ALU. Therefore, they can be added for minimal cost and improve the opportunities for spatial locality; increased spatial locality reduces crossbar activity and increases the utility of each input port to the functional unit and the peripheral logic that supports the functional unit. Furthermore, maintaining a couple universal functional units within each cluster, rather than a set of FU types without functional overlap, provides better performance for applications within the domains that do not require specialized hardware.

9. REFERENCES

[1] P. Hung, H. Fahmy, O. Mencer, and M. Flynn, "Fast division algorithm with a small lookup table," vol. 2, 1999, pp. 1465 –1468 vol.2.

[2] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *FPGA*. New York, NY, USA: ACM, 1998, pp. 191–200.

[3] B. Van Essen, A. Wood, A. Carroll, S. Friedman, R. Panda, B. Ylvisaker, C. Ebeling, and S. Hauck, "Static versus scheduled interconnect in Coarse-Grained Reconfigurable Arrays," in *FPL*, 31 2009-Sept. 2 2009, pp. 268–275.

[4] B. Van Essen, R. Panda, C. Ebeling, and S. Hauck, "Managing short versus long lived signals in Coarse-Grained Reconfigurable Arrays," in *FPL*. Milano, Italy: IEEE, Aug. 31 - Sept. 2 2010, pp. 380–387.

[5] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[6] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[7] D. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," in *Advanced Research in VLSI*, Atlanta, 1999, pp. 23–40.

[8] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource Sharing and Pipelining in Coarse-Grained Reconfigurable Architecture for Domain-Specific Optimization," in *DATE*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 12–17.

[9] S. Wilton, N. Kafafi, B. Mei, and S. Vernalde, "Interconnect architectures for modulo-scheduled coarse-grained reconfigurable arrays," in *FPT*, 6-8 Dec. 2004, pp. 33–40.

[10] H. Lange and H. Schroder, "Evaluation strategies for coarse grained reconfigurable architectures," in *FPL*, 24-26 Aug. 2005, pp. 586–589.